Ceph: A Scalable, High-Performance Distributed File System

Scott A. Brandt
Associate Director
Storage Systems Research Center
University of California, Santa Cruz

Who am I?

- **Associate Professor** and **Director of Graduate Studies**, Computer Science, UC Santa Cruz
- **Associate Director**, UCSC Storage Systems Research Center (SSRC) and UCSC/Los Alamos Institute for Scalable Scientific Data Management (ISSDM)
- **Director**, UCSC Real-Time Systems Laboratory
- Background
  - 1999 Ph.D. CS, Colorado
  - 1987/1993 B. Math/M.S. CS, Minnesota
  - 1982–1994 Programmer/Research Scientist/VP CPT, B-Tree, Honeywell SRC, Theseus Research, Alliant TechSystems RTS, Secure Computing

- Current Research
  - Storage Systems
    - High-performance peta-scale storage
    - New storage technologies
    - Enhanced metadata management
  - Real-Time Systems
    - Integrated hard real-time, soft real-time, and non-real-time processing
- Past Research
  - Real-time image processing systems
  - Secure operating systems
  - Asynchronous circuits and parallel programming languages

Peta-scale Data Storage: Our Goals

Performance

- 20 PB storage system
  - 1-10,000 hard drives
- 1 TB/sec aggregate throughput
  - 1-10,000 hard drives pumping out data as fast as they can
- Billions of files
  - Bytes to terabytes
  - 1-100,000+ files/directory
- Very low-latency metadata

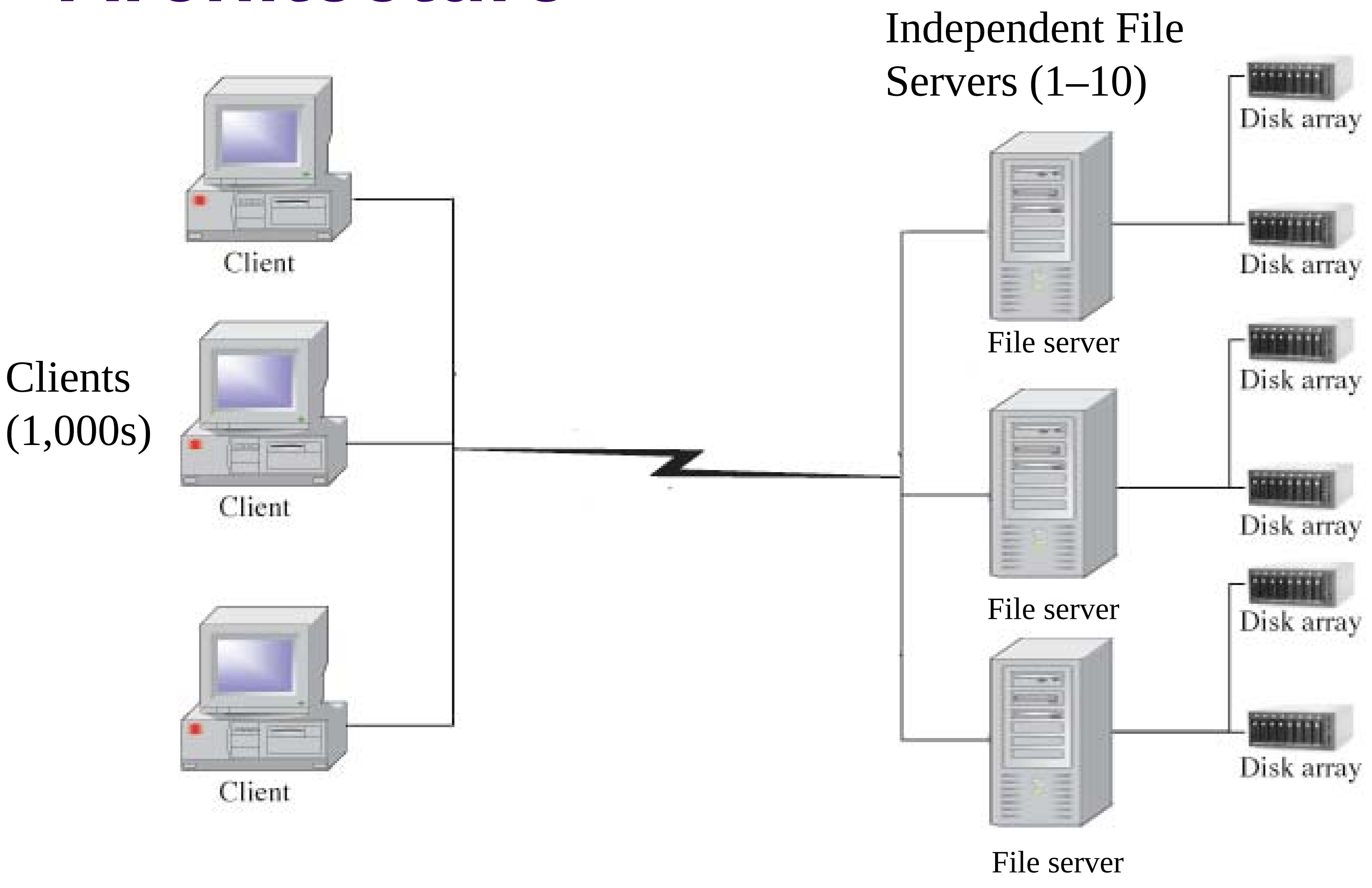
Usage

- POSIX-like interface
  - Standard file/directory semantics
- High-performance direct access from 100,000+ clients, to
  - Different directories, same directory, same file
- Mid-performance local access by visualization workstations w/QoS
- Wide-area general-purpose access

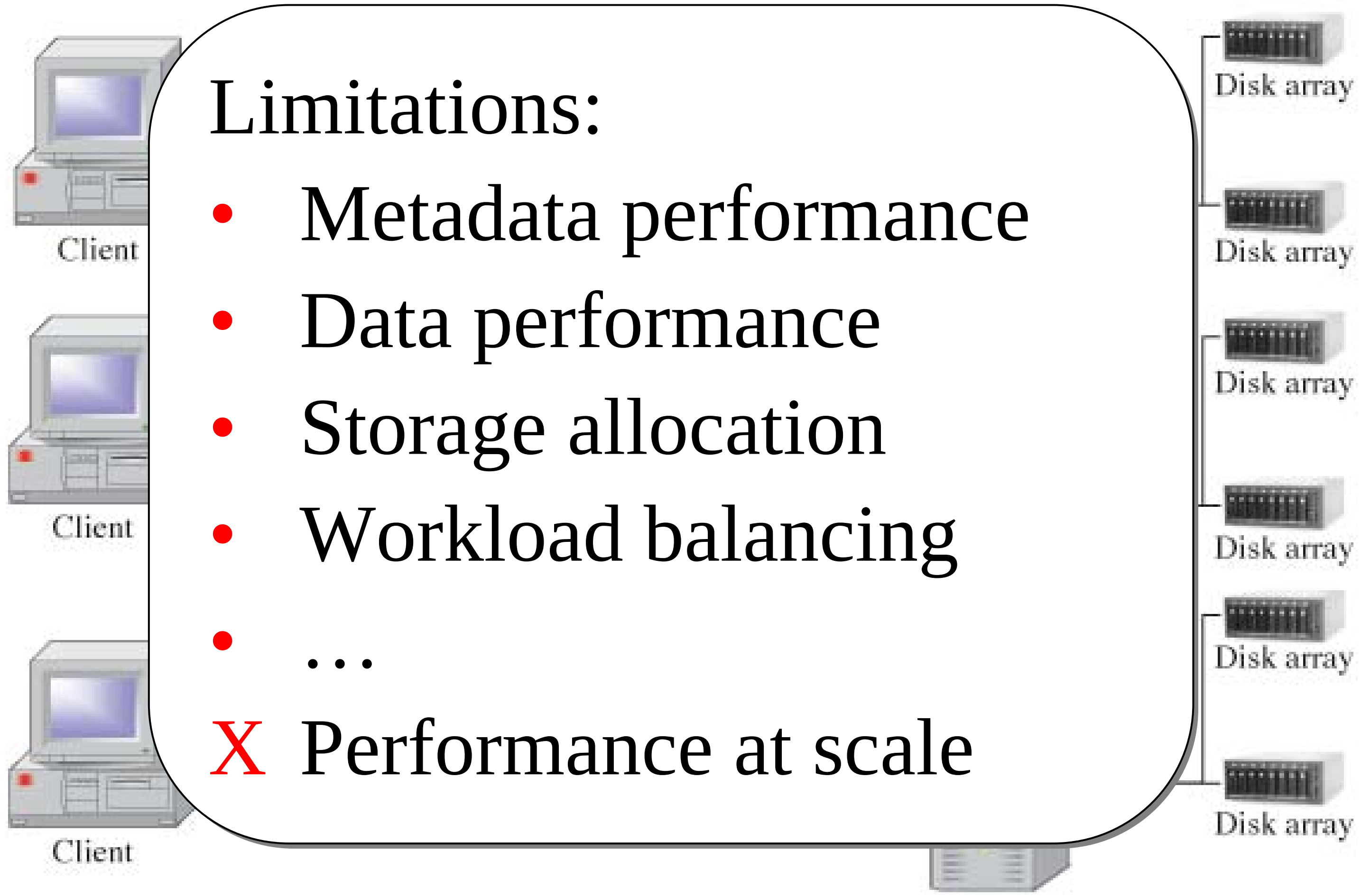
Peta-scale Data Storage Challenges

- Massive scale of everything
  - Huge files, directories, data transfers, etc.
- Managing the data
  - Coordinating the activity of thousands of disks
- Managing the metadata
  - Unified directory hierarchy
- Workload
  - Scientific and general purpose workloads
- Dynamic capacity
  - Must be able to grow (or shrink) dynamically
- Reliability
  - Thousands of hard drives $\Rightarrow$ frequent failures
- Security
  - Authentication, encryption, etc.
- Performance
  - Hot spot avoidance
  - Many possible bottlenecks
- Quality of Service
  - Guaranteed performance with mixed workloads
- Usability
  - Finding anything among all of that data

Traditional Storage System Architecture



Traditional Storage System Architecture

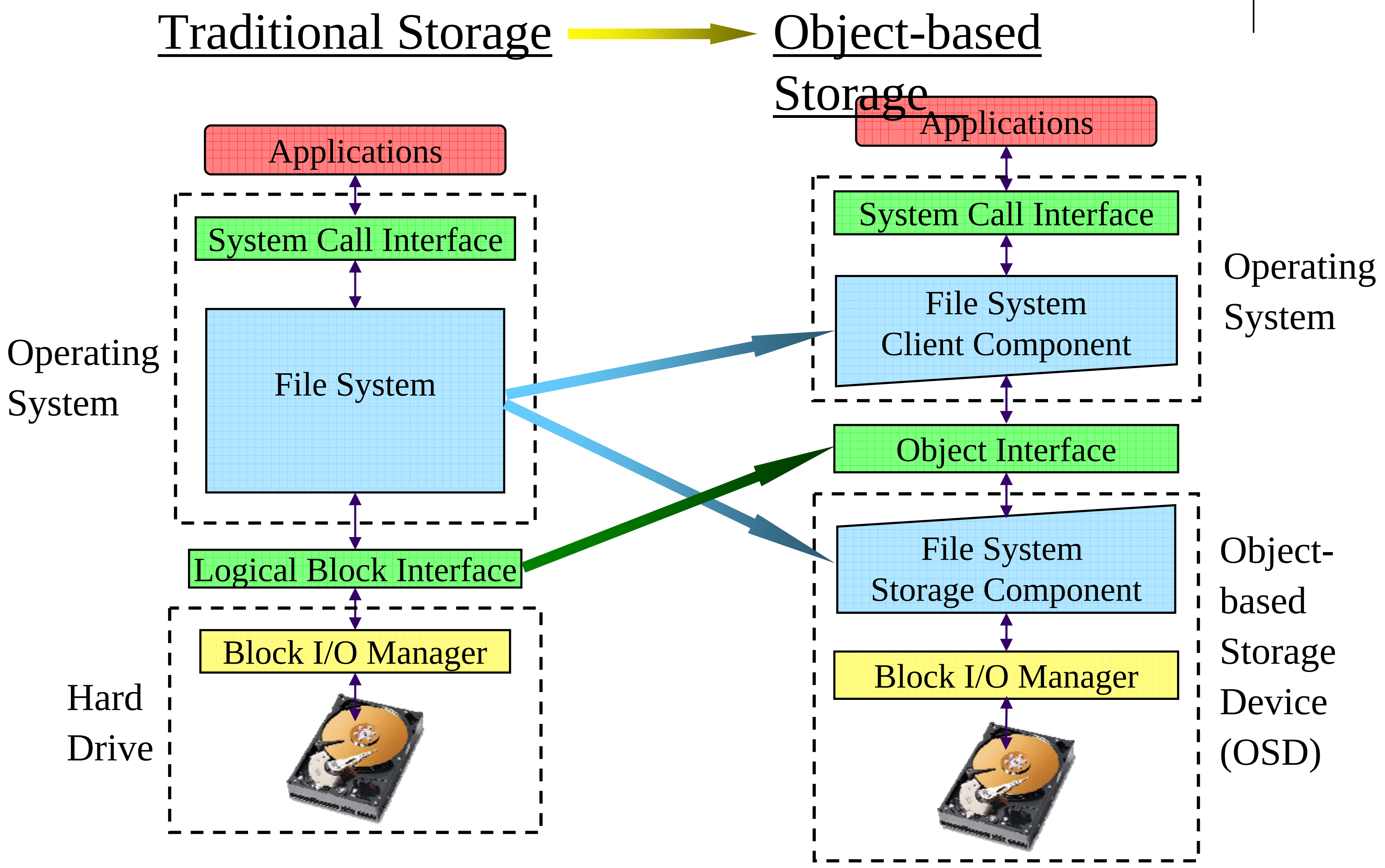


Limitations:

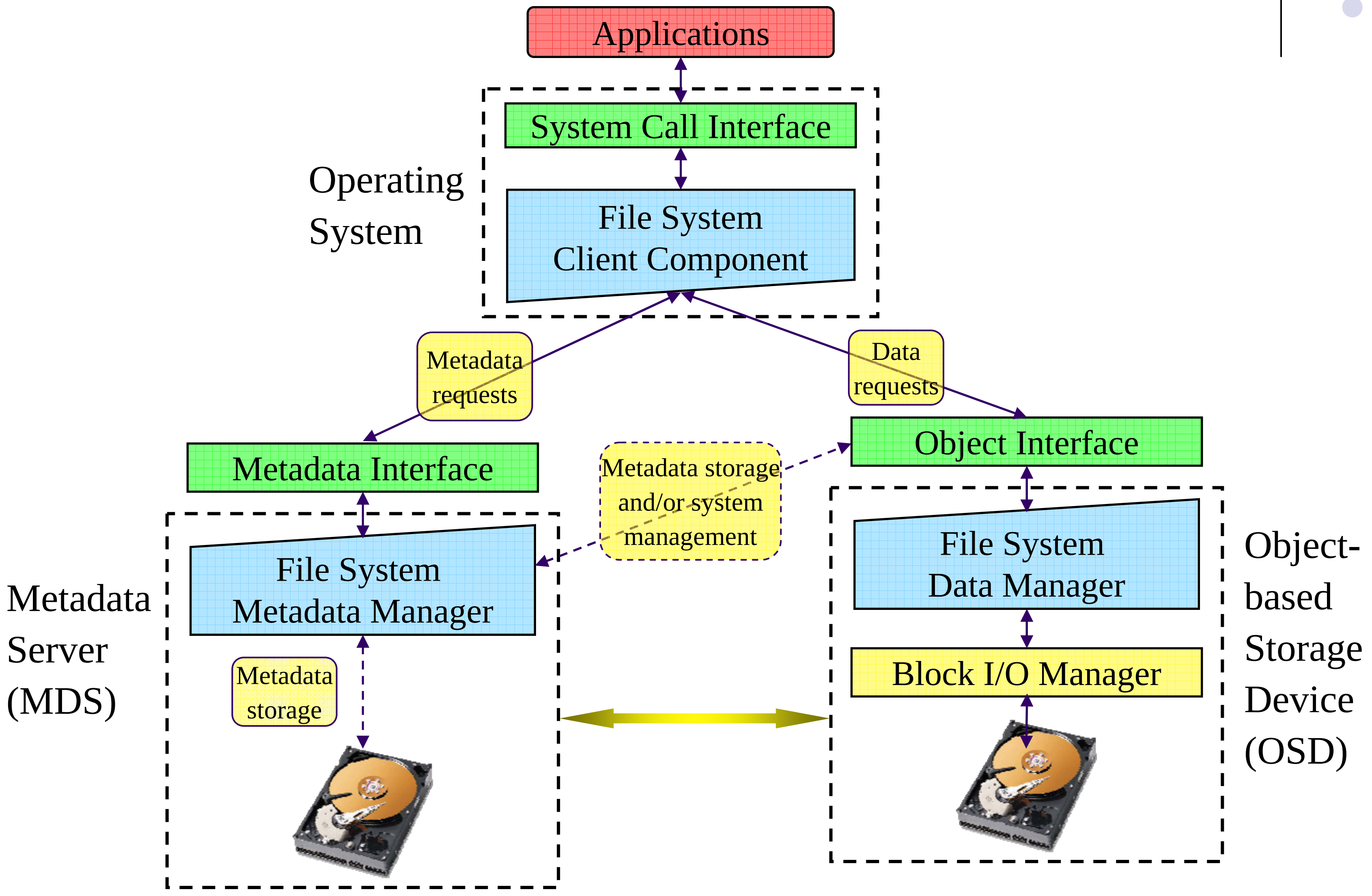
- Metadata performance
- Data performance
- Storage allocation
- Workload balancing
- …

X Performance at scale

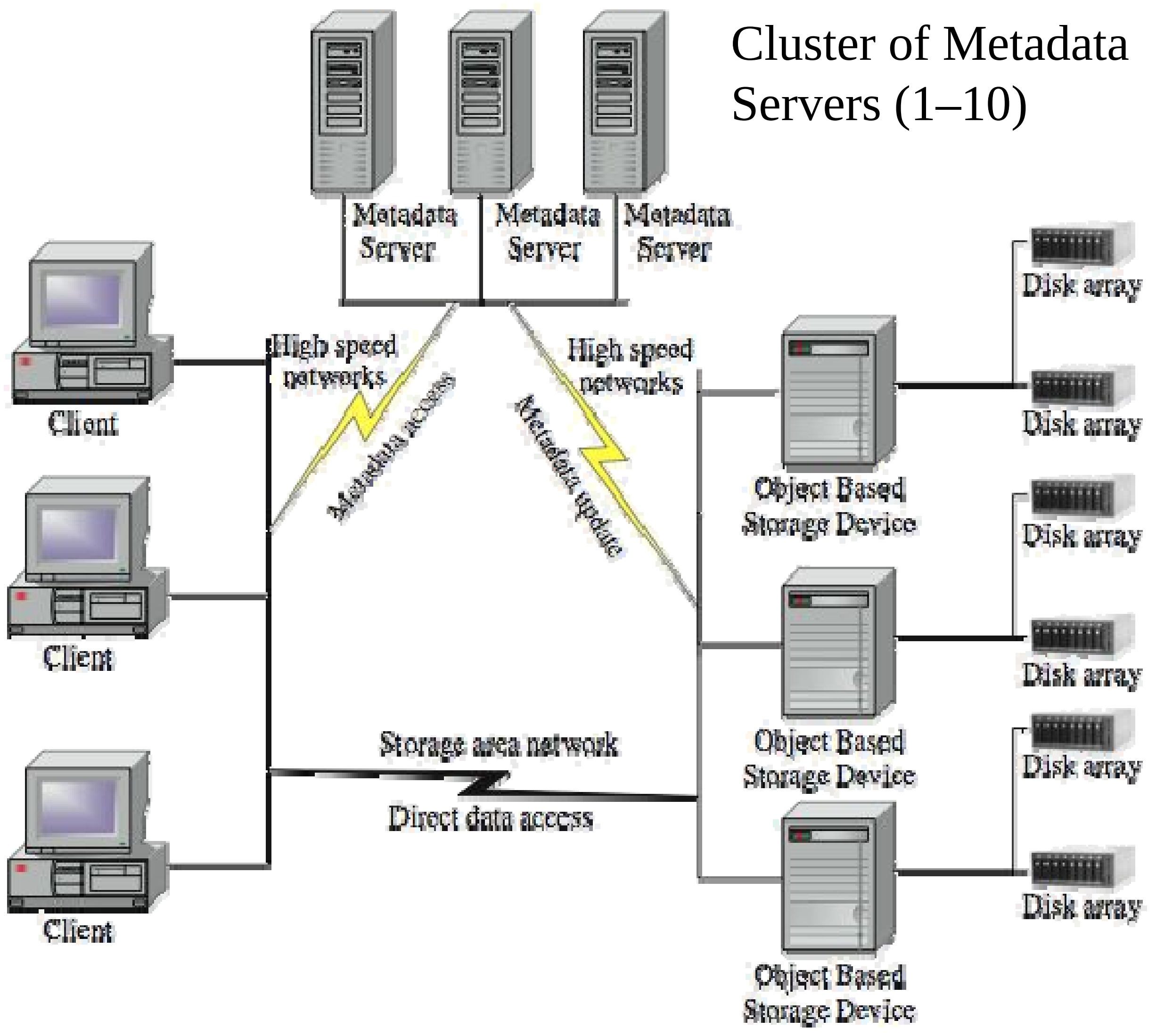
First Key Idea: Object-based Storage



Second Key Idea: Decoupled Data and Metadata



Peta-scale Object-based Storage System Architecture

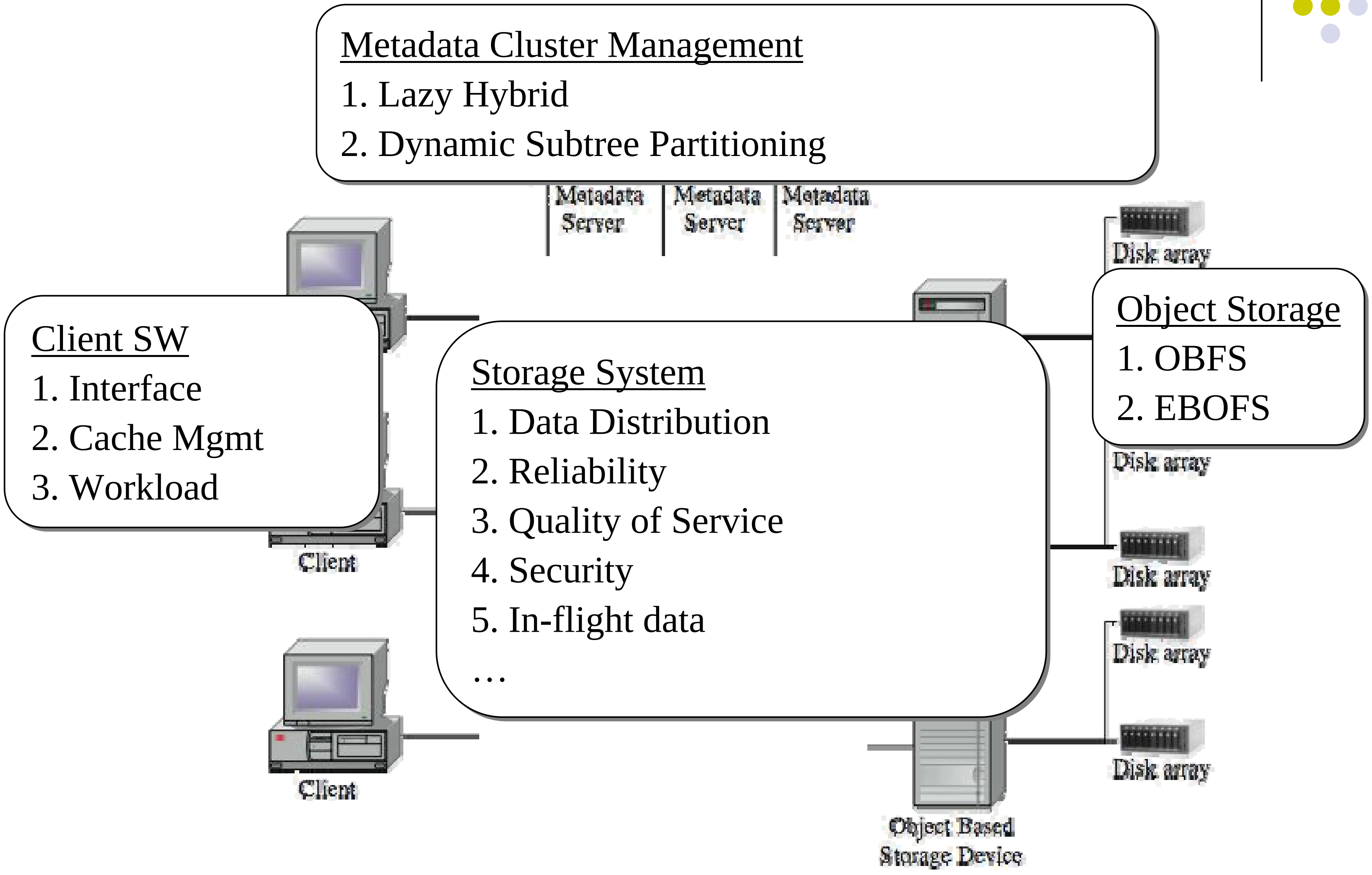


Cluster of Metadata Servers (1–10)

Clients (10,000+)

Object-based Storage Devices (1–10,000)

Our Research



Metadata Cluster Management

1. Lazy Hybrid
2. Dynamic Subtree Partitioning

Client SW

1. Interface
2. Cache Mgmt
3. Workload

Storage System

1. Data Distribution
2. Reliability
3. Quality of Service
4. Security
5. In-flight data

…

Object Storage

1. OBFS
2. EBOFS

Ceph Goals

- **Reliable**, **high-performance** distributed file system with unprecedented **scalability**
  - POSIX-like interface
  - Petabytes to exabytes, multi-terabyte files, billions of files
  - Hundreds of thousands of clients simultaneously accessing same files or directories

- Object-based storage promises scalability, but has largely failed to deliver due to continued reliance on traditional storage systems principles
  - Inode tables
  - Block (or object) list allocation metadata
  - Unintelligent storage devices

Four Key Design Principles

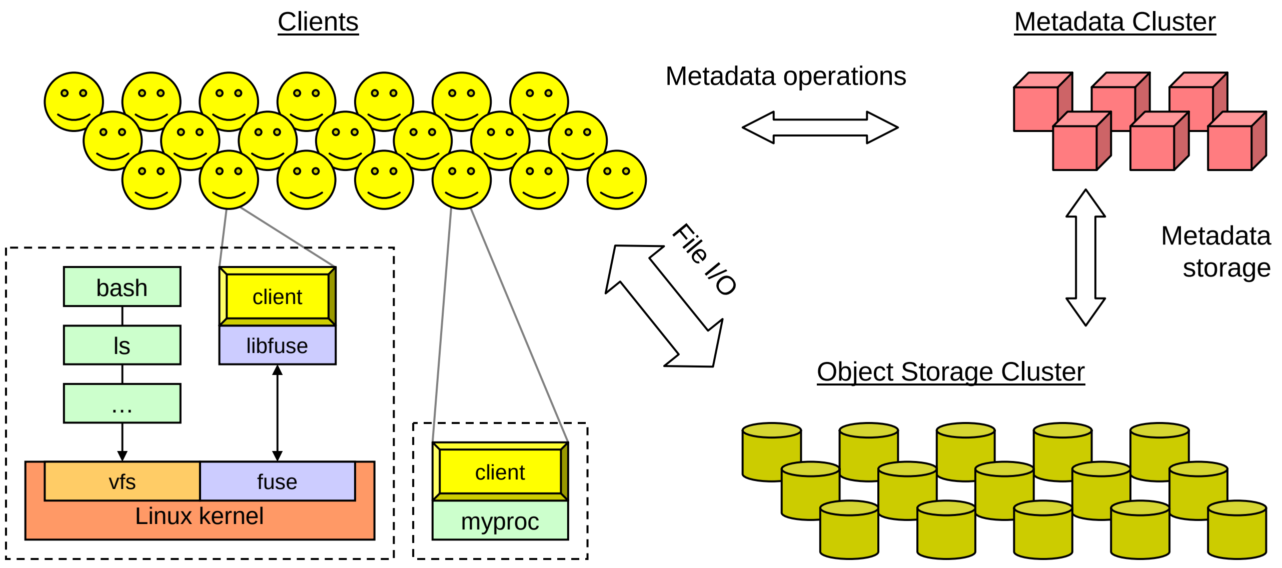
1. Separation of data and metadata
2. Pseudo-random data placement
3. Robust distributed object storage
4. Dynamic distributed metadata management

Overview

- Client operation
  - System overview, extending POSIX
- CRUSH – pseudo-random data placement
- DSP – distributed metadata
  - Traffic management, storage
- RADOS – reliable, distributed object storage
  - Intelligent OSDs, specialized local object storage
- EBOFS – high-performance object storage
- Evaluation

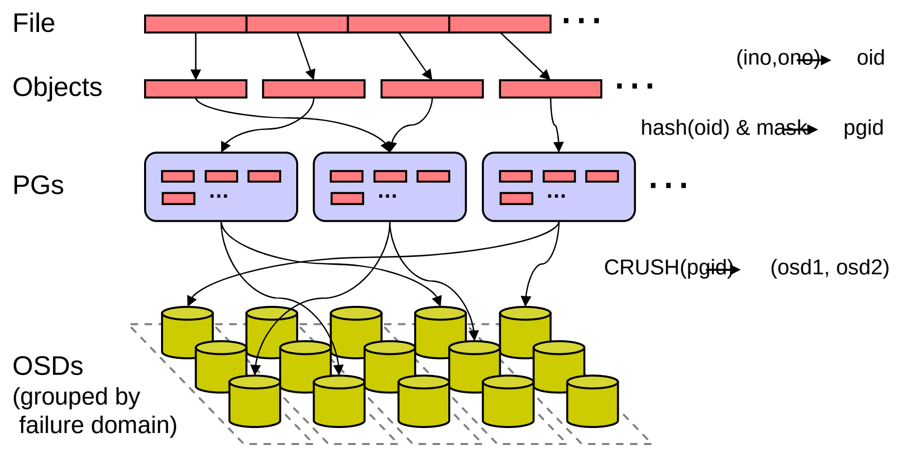
Client Operation

- Clients expose Ceph interface to a process or host
  - Near-POSIX: we extend the interface and selectively relax consistency semantics
  - Can link to a single process or mount (via FUSE)
- Decoupled data and metadata operations



Client Operation (cont.)

- Client sends *open* request to MDS
  - Receives a **capability**, granting client permission to read or write to objects comprising file
  - Also receives inode number, striping information
- Client reads/writes directly to OSDs
  - Maps file contents onto objects based on striping strategy
  - Generates object names using inode and object number
  - Calculates object locations using CRUSH function



Extending POSIX—Lazy I/O

- Mixed readers/writer or multiple writers shifts clients to synchronous I/O mode
  - Updates serialized at OSDs for proper semantics
  - Increases latency—can kill performance!
- Ceph implements subset of proposed HPC I/O extension
  - O_LAZY option for *open()* relaxes consistency when applications opt to manage it themselves
  - *lazyio_propogate*(), *lazyio_synchronize*() allow application to force updates to be visible to others
- Retains simplicity and enables high-performance without breaking consistency system-wide (as NFS3 does)

Extending POSIX— *readdir() + stat()*

- Common, and slow—typically involves many lookups in inode table
- Ceph MDS embeds inodes in directories
  - A single OSD access fetches directory contents *and* inodes into MDS cache
  - A client *readdir()* retrieves directory entries and inode contents with a single request to an MDS
- *readdirplus()* system call provides applications with appropriate semantics
  - Or, Ceph can relax consistency for a *stat()* immediately following a *readdir()*

CRUSH—
Robust Data Distribution

- Controlled Replication Under Scalable Hashing
  - Pseudo-random replica distribution algorithm
  - No allocation metadata (no lookups in data path!)
  - Efficiently reorganizes data when the cluster changes due to addition or removal of storage
  - Enforces flexible constraints on replica distribution to enhance reliability – failure domains

CRUSH is a *function*

CRUSH(x) → (osd4, osd21, osd13)

- CRUSH maps an integer identifier *x* to an ordered list of storage targets
  - No lookup tables
  - No block or object lists associated with each file
    - *x* is just an integer – we use *psid ← hash(object_id) & mak*
  - Pseudo-random – looks random, but deterministic!

- Implicit inputs include
  - A ***hierarchical cluster map*** describing available storage devices
  - A ***placement rule*** describing any constraints on object placement
    - How many replicas
    - Separation of replicas across failure domains

- Everybody has the cluster map and placement rules
  - ***Calculate*** object locations instead of looking them up

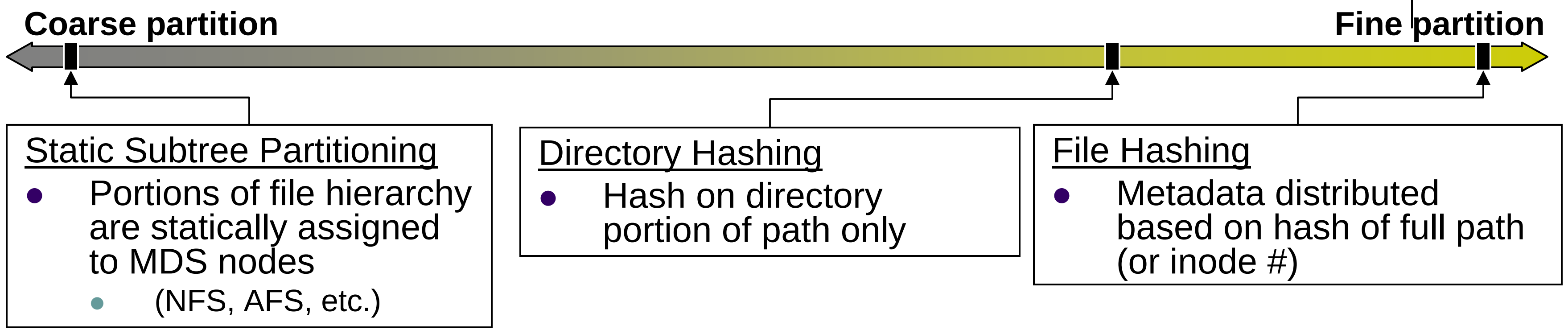
CRUSH maximizes reliability, minimizes data migration

- Cluster map represents OSDs as a hierarchy that reflects arbitrary *physical* or *logical* structure
  - Shelves, cabinets, rows, rooms, buildings
  - Power supplies, networks, racks
  - Performance, reliability, …
- Placement rules define replica placement behavior
  - *e.g.* 3 replicas, in same row, but each in a different cabinet
- CRUSH mapping is **stable**
  - When disks are added, removed, or fail, CRUSH minimizes the amount of data that migrates to maintain balance

DSP: Dynamic Distributed Metadata Management

- Dynamic Subtree Partitioning
- Workload Partitioning
  - Distribute MDS workload
- Traffic Management
  - Coping with hot spots
  - Directing client traffic
- Metadata Storage
  - Fast commits and efficient reads
  - Data safety and MDS failure recovery

Traditional Partitioning



Static Subtree Partitioning

- Portions of file hierarchy are statically assigned to MDS nodes
  - (NFS, AFS, etc.)

Directory Hashing

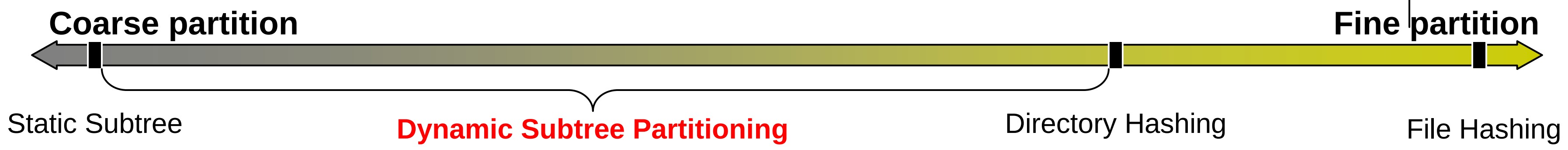
- Hash on directory portion of path only

File Hashing

- Metadata distributed based on hash of full path (or inode #)

- Coarse distribution (static subtree partitioning)
  - hierarchical partition preserves locality
  - high management overhead: distribution becomes imbalanced as file system, workload change
- Finer distribution (hash-based partitioning)
  - probabilistically less vulnerable to “hot spots,” workload change
  - destroys locality (ignores underlying hierarchical structure)

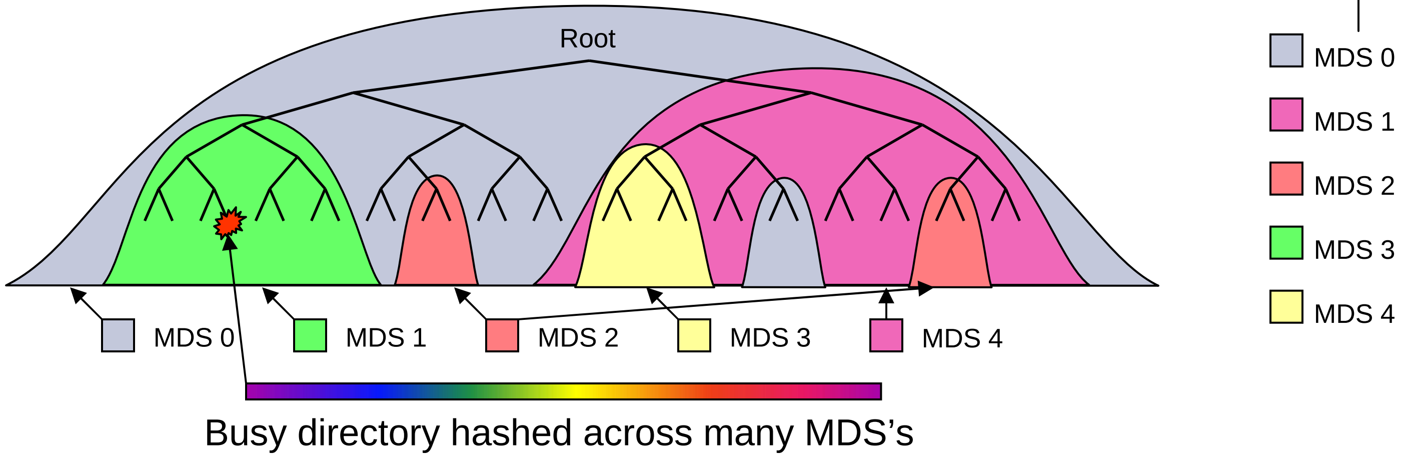
Ceph’s Dynamic Partitioning



- Distribute subtrees of directory hierarchy
  - Somewhat coarse distribution of variably-sized subtrees
  - Preserve locality within branches of the directory hierarchy

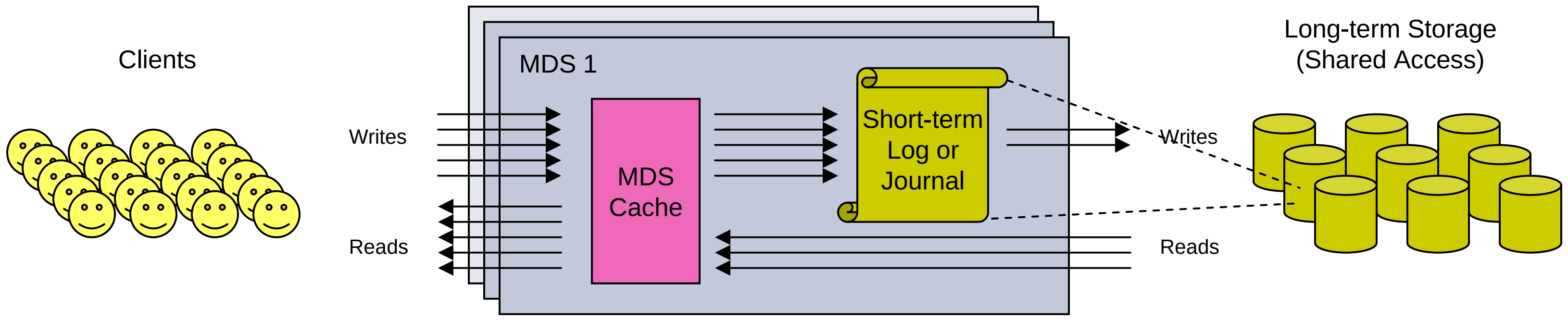
- Intelligently manage distribution based on workload demands
  - Keep MDS cluster load balanced
  - Actively repartition as workload and file system change instead of relying on a (fixed) probabilistic distribution

Metadata Partition



- Subtrees are dynamically redistributed to balance workload
  - Granularity ranges from large subtrees to individual directories
  - Coarse partition preserves locality, improves efficiency
- Ceph adapts to hotspots in workload
  - Heavily read directories are replicated on multiple MDSs
  - Heavily written directories are **hashed** across the entire cluster

Metadata Storage— Two Tiers



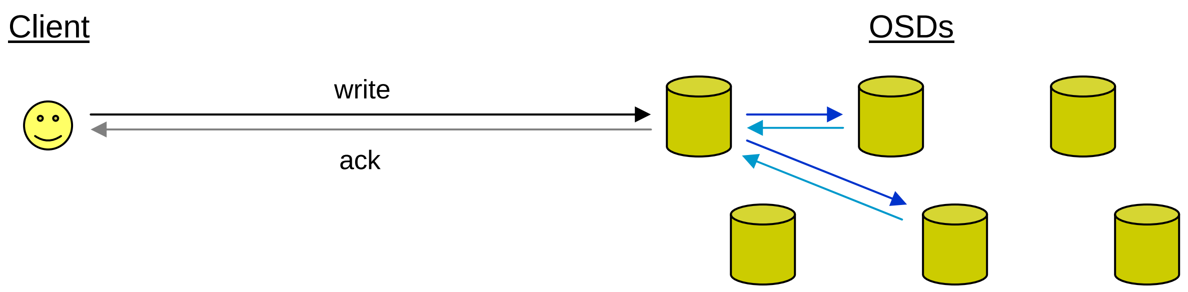
- Short-term storage in metadata journal
  - Immediate commits require **high sequential write bandwidth**
  - Very large journal with extremely lazy commits
  - Absorbs short-lived or repetitive metadata updates
  - Used for recovery after MDS failures
- Long-term storage
  - On-disk layout **optimized for future read access**
  - Inodes embedded in directories—no large, awkward inode tables

RADOS—Reliable Autonomic Distributed Object Store

- OSDs self-managing, distribute
  - replication
  - failure detection (refereed by third party)
  - failure recovery
  - data migration
- Single object namespace
  - Individual OSDs store different objects, but
  - Cluster collectively acts and appears as a single, reliable, self-managing distributed store

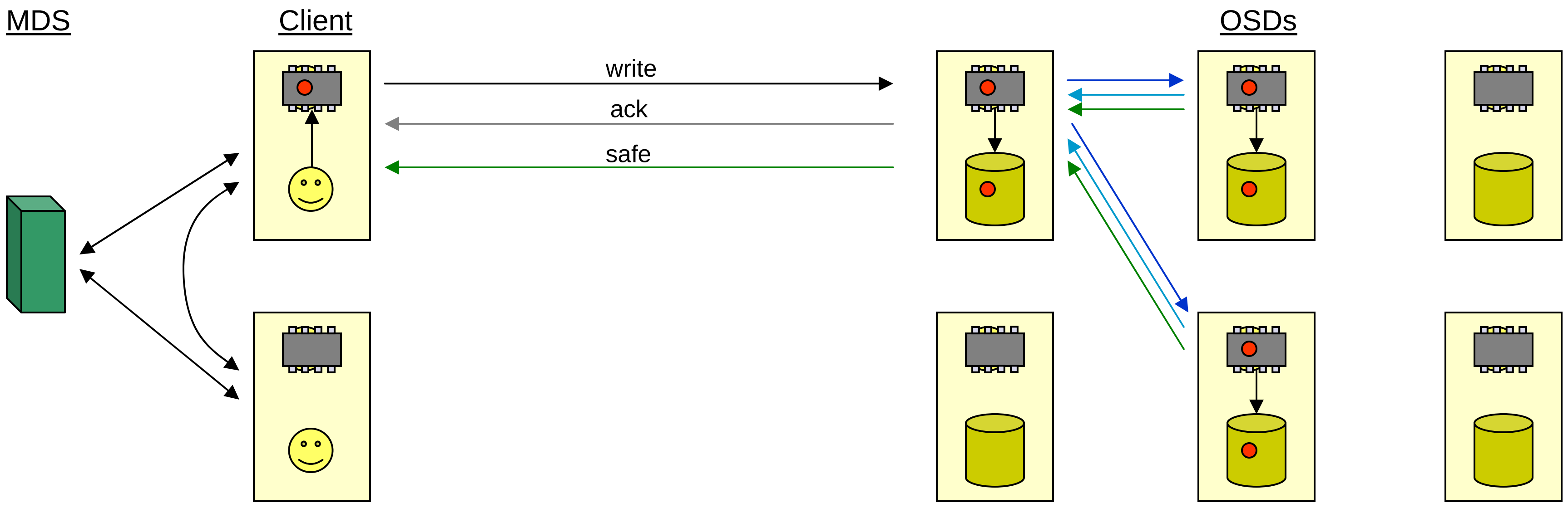
RADOS – Replication

- Clients send reads, writes to first OSD
  - Reads are satisfied locally
    - or delegated to replicas for fine-grained load balancing
  - Writes are forwarded to replica sites
    - Ack'ed only after replicas ack
    - Leverages local OSD interconnect, intelligence



RADOS – Synchronization vs Safety

- Two reasons we write data to the object store
  - Synchronization – so other clients can see it
    - Must be *fast* – maintain consistency and coherency without killing performance
  - Safety – data on disk survives power failures, etc.
    - Must be *reliable* – either assumed, or often because the application asks for it with fsync()
- RADOS disassociates write ***ack*** and ***safe*** replies
  - Client receives quick **ack** when write is received by object store and applied to replica buffer cache(s)
  - A second **safe** follows (seconds?) later when data is safe on disk



RADOS – Failure Detection

- OSD Failure detection is distributed
  - Each OSD monitors a subset of its peers via ping or heartbeat messages
  - Piggybacks on existing inter-OSD replication chatter when possible
  - Failure reports sent to third party referee (MDS)
- Referee confirms failures
  - Filters out bogus reports, spurious connectivity failures, partitions, etc.
  - Distributes new *cluster map* with new OSD state decrees
  - Operations pending with newly-failed devices are rerouted

RADOS – Failure Recovery

- OSDs scan their current PGs
  - Any “stray” data in affected PGs is announced to the PG’s new primary OSD
  - Primary collects PG content summaries (object lists) and distributes to all replica OSDs
  - Each OSD, armed with “correct” PG contents, will independently retrieve any object replicas it is missing
- Normal workload is mostly unaffected
  - Recovery proceeds in the background
  - Updates to non-replicated data currently block while OSDs replicate underlying objects

EBOFS – Low-level object storage

- **E**xtent/**B**Tree-based **O**bject **F**ile **S**ystem
  - Extent-based allocation—(start,length) instead of block lists
  - Robust generalized BTree storage service
    - Object onode table
    - Collections (placement groups)
    - Free extent lists, indexed by size, position
  - Attributes – on objects and collections
  - Safety – efficient copy-on-write for data and metadata

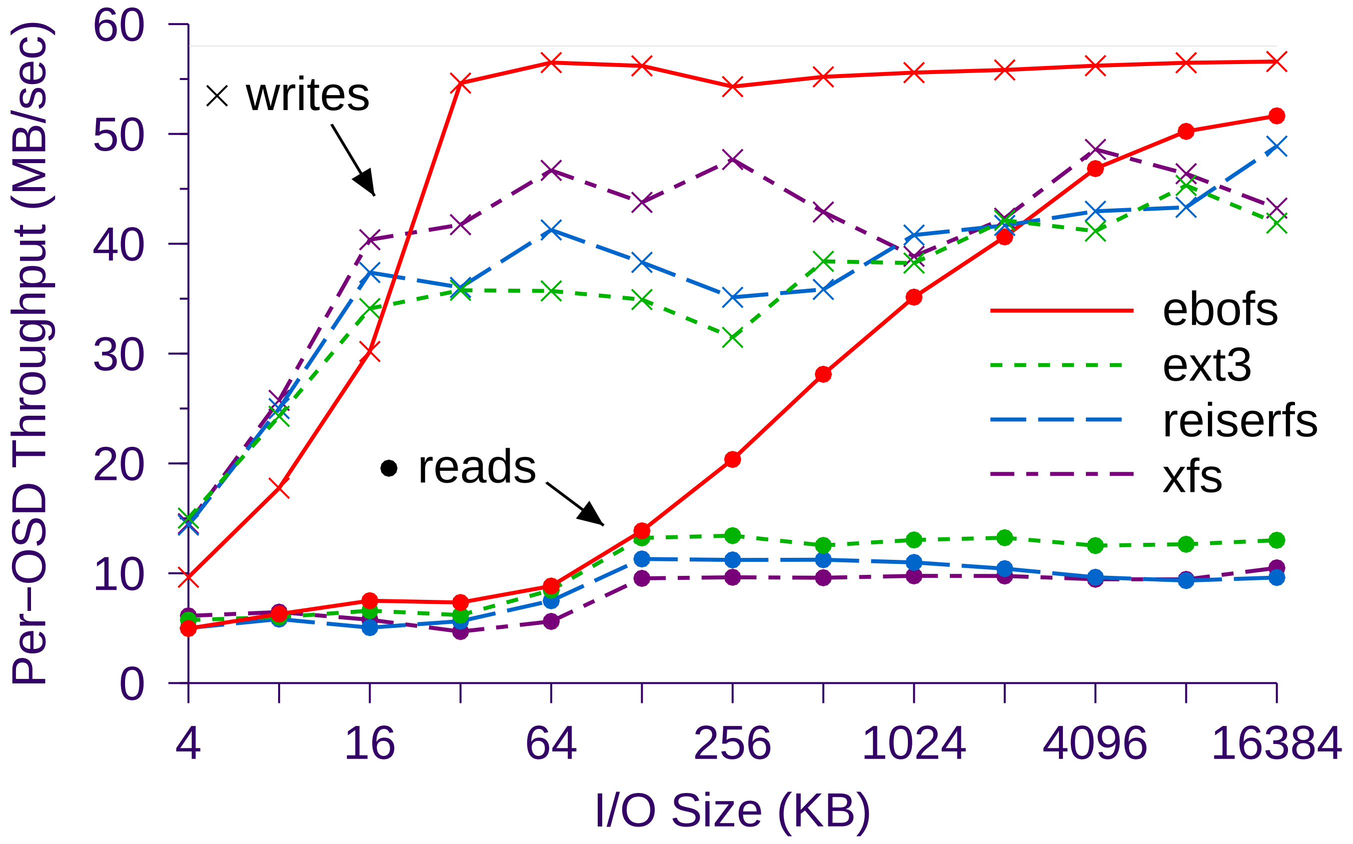
EBOFS—
Not a typical file-system!

- Non-standard transaction semantics (for RADOS)
  - Atomic compound transactions—data + metadata updates
    - Multiple writes, attribute or collection membership updates
  - Asynchronous (“safe”) notification of disk sync
- User-space implementation
  - We define our own interface
    - not limited by existing (and ill-suited) kernel POSIX interface, Linux page cache, etc.
  - Superior performance compared to general-purpose kernel filesystems (ext3, xfs, etc)

EBOFS – Emphasis on Safety

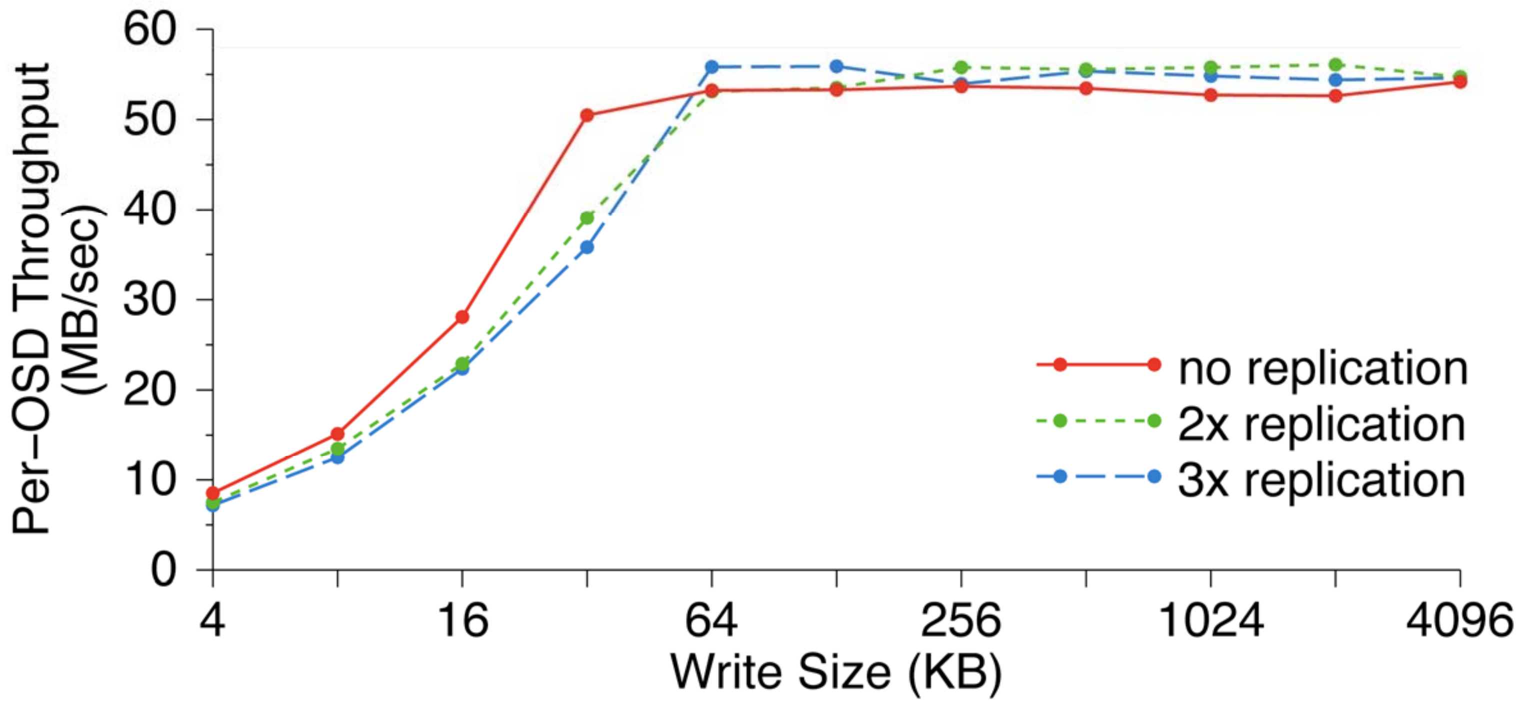
- Two copies (even and odd) of superblock
  - Alternating updates
  - Point to BTrees containing all metadata
- Copy-on-write – *all writes are to unallocated space*
  - Modified BTree nodes written to unused blocks
  - Modified data written to unallocated space
- Commit cycle every second or so
  - Dirty object data, BTree data flushed to disk
  - New superblock written
  - Asynchronous update commit callbacks triggered
  - Journaling soon?
  - Non-blocking—no effect on disk workload
- Any power failure returns EBOFS volume to fully consistent state – no fsck

OSD Performance—EBOFS vs ext3, XFS, ReiserFS



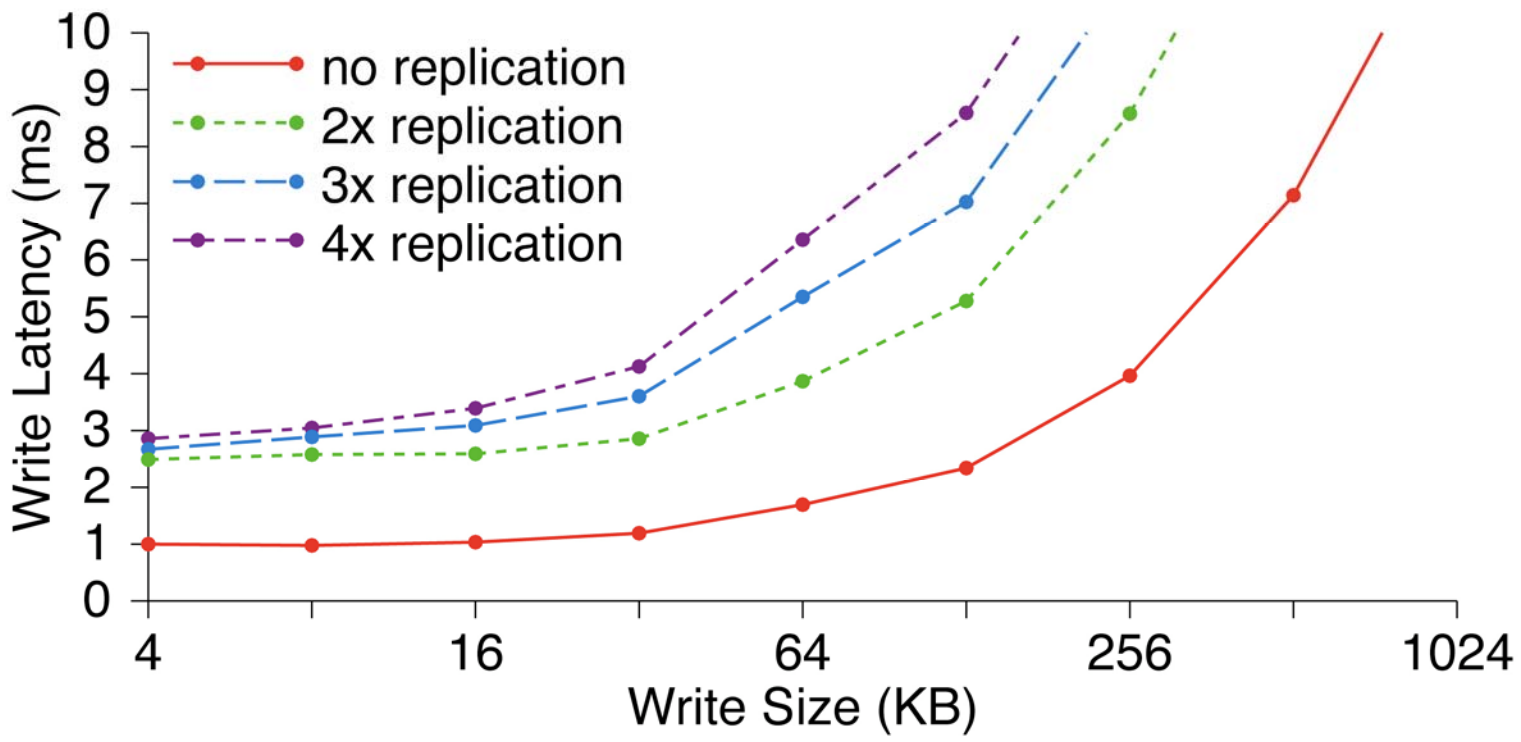
- EBOFS writes saturate disk
- Reads approach optimal when data is written in large increments

OSD Performance— Throughput w/Replication



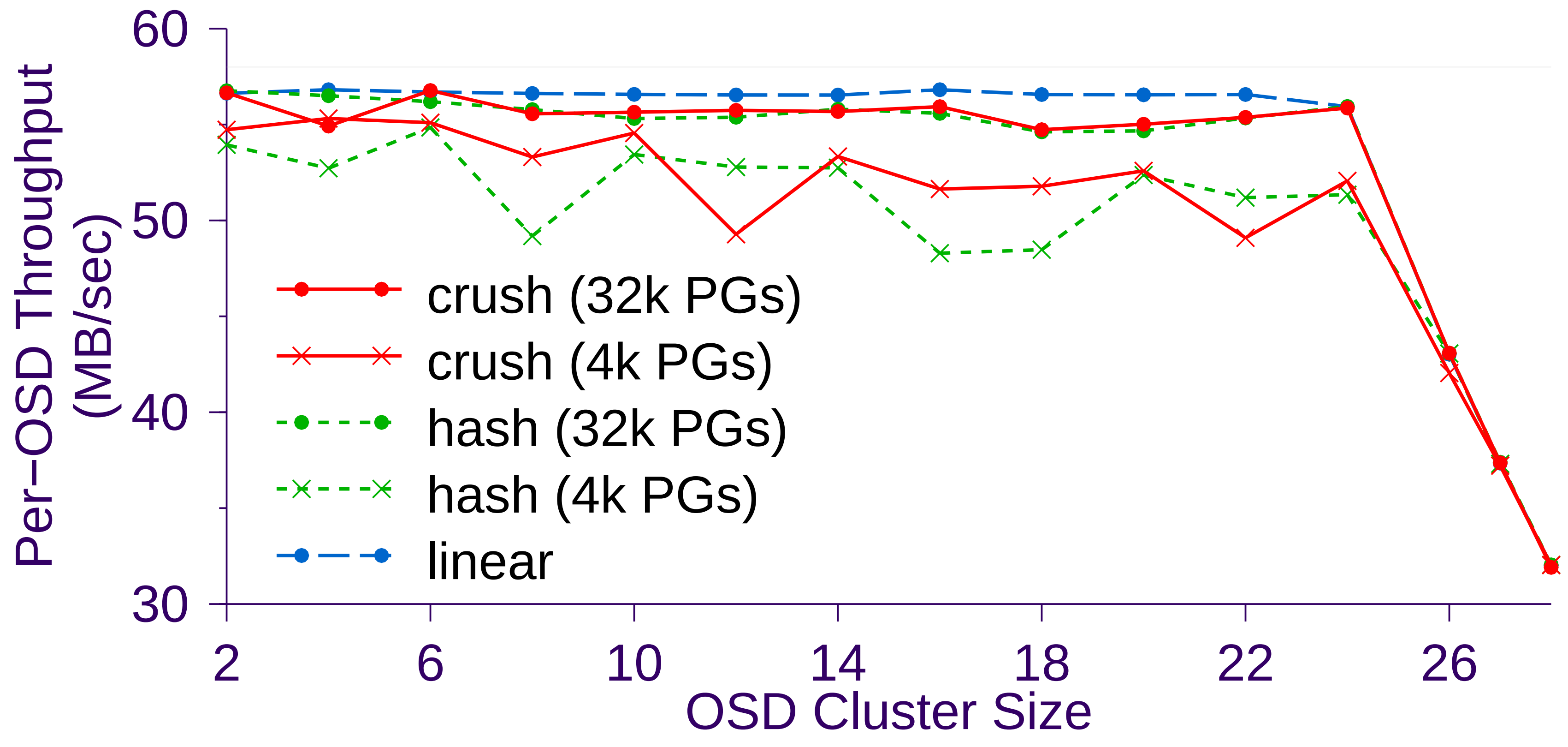
- Little per-OSD impact
- Proportional decrease in overall throughput (not seen here)

OSD Performance— Write Latency w/Replication



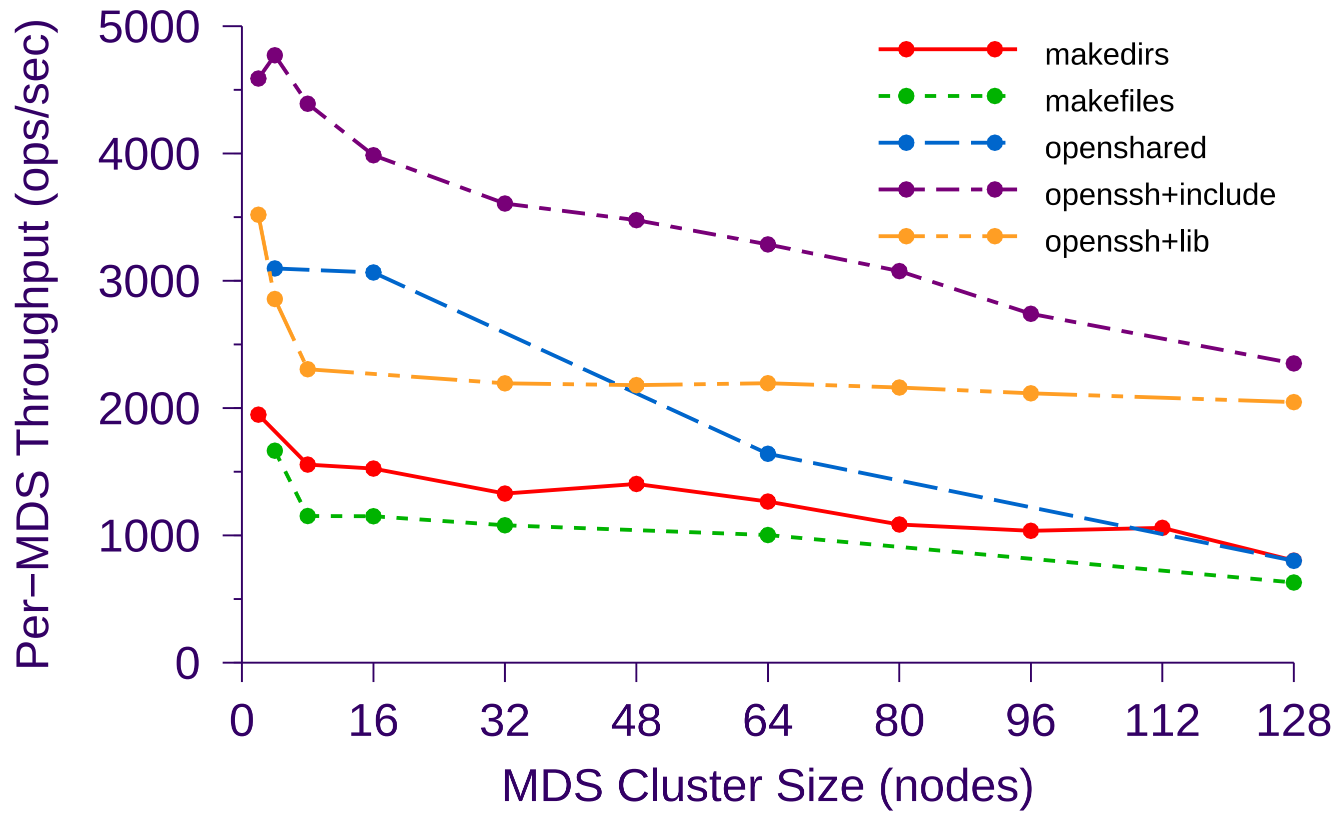
- Tolerable increase in latency w/replication
- Little extra cost for 3+x replication (performed concurrently)

OSD Cluster Scaling—CRUSH vs careful striping



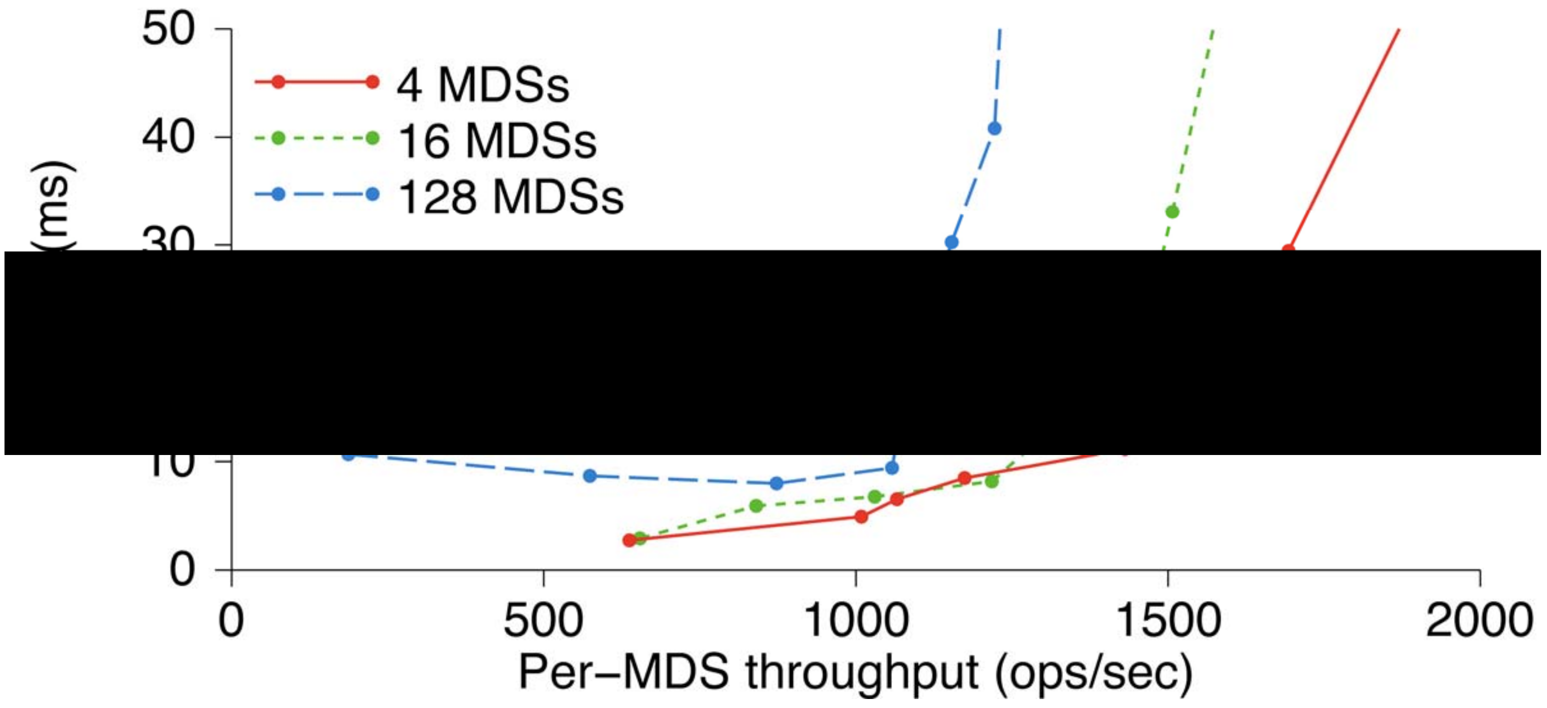
- Higher placement group count reduces statistical variance, divergence from optimal (write throughput shown)

Metadata Scalability—Throughput vs. Cluster Size



- Over 250,000 metadata ops/second!
- Potentially many terabytes/second, petabytes to exabytes!

Metadata Scalability—Latency vs. Cluster Size



[redacted]

- Good latency until saturation
- Larger cluster saturates at somewhat lower per-MDS workloads due to between-MDS communications

Conclusions

- Reliable, high-performance storage with unprecedented scalability
- Very soon: http://sourceforge.net/projects/ceph
- Help us build one (or more)!

Future Work

- Ceph QoS architecture
  - Distributed reservations and performance guarantees
- Archival storage
  - Managing data hot spots, idle data
- Rich metadata
  - Our MDS built around 30-year old POSIX file system interface
  - Next generation file systems will likely diverge from a single hierarchy
- In-flight data management
  - Better awareness (and exploitation) of data residing in transit or in client caches
  - Leverage existing object-based interface and replica synchronization techniques
- Enhanced Interfaces (maybe not us)

Thanks!

Contributors

Sage Weil

Feng Wang, Chris Xin, Lan Xue

Ethan Miller, Carlos Maltzahn, Darrell Long

Supporters

Lawrence Livermore National Laboratory

Los Alamos National Laboratory

Sandia National Laboratory